Week 2 Notes

Lectures 3 & 4

University of Massachusetts Amherst, CS389

1 Neural Networks (lecture 3)

Definition 1.1 (Multi-dimensional data)**.** A *multi-dimensional data* that has n number of data with d number of features can be written as[1]:

$$X = \left[X^{(1)}, X^{(2)}, ..., X^{(n)}\right]^T \tag{1.1}$$

where $X^{(i)}$ is the i^{th} data[2]. A single data, $X^{(i)}$ is a d-dimensional vector which has d number of features. For example, the features for a weather data may include humidity, temperature, air speed, etc. Every data is represented as a row and each feature as a column. Formally we write a single data as:

$$X^{(i)} = \left[X_1^{(i)}, X_2^{(i)}, ..., X_d^{(i)}\right] \tag{1.2}$$

Putting all this together, our multi dimensional data is a matrix with dimensions $(n \times d)$:

$$X = \begin{bmatrix} X_1^{(1)} & \cdots & X_d^{(1)} \\ \vdots & \ddots & \vdots \\ X_1^{(n)} & \cdots & X_d^{(n)} \end{bmatrix} \tag{1.3}$$

1.1 Perceptron

Definition 1.2 (Perceptron)**.** A *perceptron* is a computational model of a biological neuron. A graphical visualization of a perceptron can be seen in Figure 1,

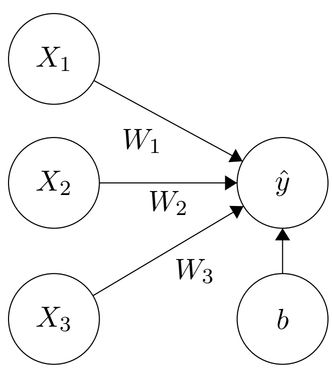

Figure 1: A single-layer perceptron

where the vertices X_i are the input features, the edges W_i are the weighted connections, the vertex b is the bias term, and the vertex Y' is the output. We can write this mathematically as:

$$\hat{y} = \phi(W^T X + b) \tag{1.4}$$

where ϕ is a non-linearity function, also known as an activation function (topic of next week).

[1] Note that this array is transposed because X is a row vector and each $X^{(i)}$ is a d-dimensional column vector

[2] The reason why we use the $X^{(i)}$ notation is because $X^{(i)}$ is a vector and sometimes we need to refer to the elements of that vector [2]

1.2 Classification

In a classification problem, we want to learn to predict discrete classes which the input belongs to.

Definition 1.3 (Binary classification)**.** Is a supervised learning algorithm that *categorizes the data into one of two classes.*

Recall that a linear model is defined as:

$$\hat{y} = \phi\left(\sum_{j=1}^{m} w_j \cdot X_j + b\right) = \phi(W^T X + b) \tag{1.5}$$

where the output $\hat{y} \in \mathbb{R}$ and we can apply a type of boolean function (i.e. the *sign* activation function) so that the output $\hat{y} \in \{-1, +1\}$. Given a fixed model, the binary classifier is defined as:

$$\hat{y} = \text{sign}(W^T X + b) = \begin{cases} +1 & \text{if } W^T X + b > 0 \\ -1 & \text{otherwise} \end{cases} \tag{1.6}$$

As can be seen in Figure 2, the points that are on the same side as the normal vector to the hyperplane is the positive half-spaces are classified as positive. Likewise, the other half-space is negative and all points are classified as negative [2].

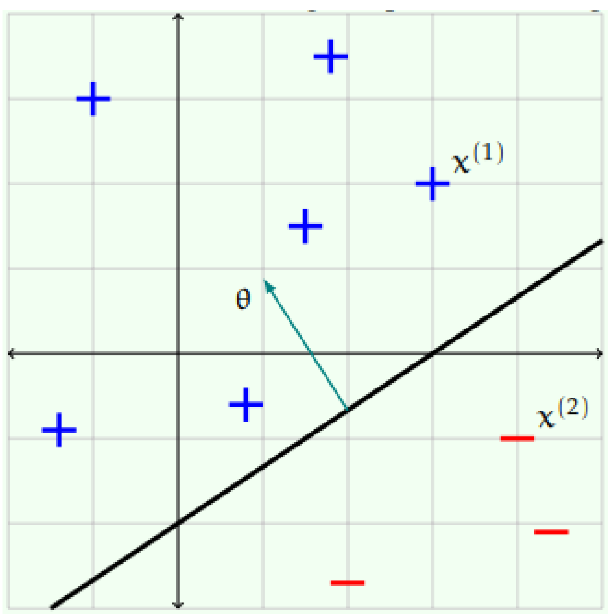

Figure 2: Binary Classification Example. The θ is the same variable as W [2]

1.3 Linear Separability

Theorem 1. *If the dataset, D, is linearly separable, then the perceptron algorithm is guaranteed to find a linear separator* [2]

How would one *formally describe linear separability?* This is beyond the scope for this class, but you can refer to this MIT lecture notes (pg. 4) for a detailed explanation. The intuition is if the shortest distance of a point to the hyperplane (the norm) is positive for all points then the dataset is classified correctly. You can use Figure 2 to help with this visualization. Understanding this helps us understand why the perceptron is unable to solve XOR.

1.4 Regression

Definition 1.4 (Regression)**.** Predict continuous outputs ($\hat{y} \in \mathbb{R}$) that are close to the true values

2 Stochastic Gradient Descent implementation (lecture 4)

Hopefully, we now have a mathematical understanding and an intuition of all the main components of a supervised machine learning model. We can now start implementing a very simple SGD. Recall that in a SGD, we want to update the parameters W and b after every single training data.

Algorithm 1 Stochastic Gradient Descent algorithm

for e in range Epoch **do** ▷ number of epochs
  for $X^{(i)}$ in X **do** ▷ loop through entire dataset
    $\hat{y}^{(i)} = X^{(i)}W^T + b$ ▷ our model's prediction for input $X^{(i)}$
    $\text{Loss}^{(i)} = \frac{1}{2}\sum(y^{(i)} - \hat{y}^{(i)})^2$ ▷ calculate MSE loss of prediction with actual
    $W = W - \alpha\frac{\partial L^{(i)}}{\partial W}$ ▷ update weight
    $b = b - \alpha\frac{\partial L^{(i)}}{\partial b}$ ▷ update bias
  end for
end for

2.1 Calculating Gradient of Loss

The gradient of loss can be written as $\nabla_W\text{Loss}$ or $\frac{\partial L}{\partial W}$. The gradient of loss is a vector of partial derivatives. We will often use terms such as "the gradient on x" instead of the technically correct phrase "the partial derivative on x" for simplicity [3]:

$$\nabla_W\text{Loss} = \frac{\partial L}{\partial W} = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, ..., \frac{\partial L}{\partial w_m}\right) \tag{2.1}$$

Using the chain rule we have:

$$\frac{\partial L}{\partial W_j} = \frac{\partial \hat{y}}{\partial W_j}\frac{\partial L}{\partial \hat{y}} \tag{2.2}$$

Now recall that $\hat{y} = W^TX + b$ and $L = \frac{1}{2}\sum(y^{(i)} - \hat{y}^{(i)})^2$, therefore:

$$\frac{\partial \hat{y}}{\partial W_j} = X_j^{(i)} \tag{2.3}$$

$$\frac{\partial \hat{L}}{\partial \hat{y}} = \hat{y}^{(i)} - y^{(i)} \tag{2.4}$$

$$\nabla_W\text{Loss} = \frac{\partial L}{\partial W} = \left[X_1^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)}), X_2^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)}), ..., X_n^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)})\right] \tag{2.5}$$

$$\therefore \frac{\partial L}{\partial W} = \left[X_1^{(i)}, X_2^{(i)}, ..., X_n^{(i)}\right] \cdot (\hat{y}^{(i)} - y^{(i)}) = X^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)}) \tag{2.6}$$

References

[1] Cooper.

[2] MIT Open Learning Library, 6.036, Spring 2020, https://openlearninglibrary.mit.edu/assets/courseware/v1/2481f8f2964716032b134db99e369b81/asset-v1:MITx+6.036+1T2019+type@asset+block/notes_chapter_Introduction.pdf

[3] Fei-Fei Li, Jiajun Wu, and Ruohan Gao, Stanford CS231n, Spring 2022. https://web.archive.org/web/20230109135558/https://cs231n.github.io/